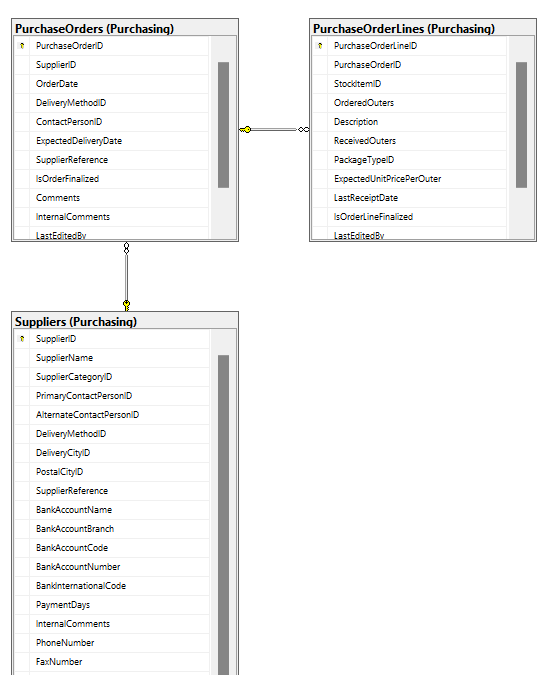


# TOP QUERY (1)

## 

## USE: WideWorldimporters Database



PROPOSITION: Create or update a view named Purchasing.SupplierPerformanceStats to store summarized performance metrics for suppliers based on their historical

# Columns from their respective tables in the select clause:

| Table Name | Column Name |
| --- | --- |
| Purchasing.Suppliers,, | SupplierID, SupplierName |
| Purchasing.PurchaseOrders | SupplierID (used for JOIN with Purchasing.Suppliers)  OrderDate  PurchaseOrderID |
| Purchasing.PurchaseOrderLines | PurchaseOrderID (used for JOIN with Purchasing.PurchaseOrders)  ExpectedUnitPricePerOuter  OrderedOuters |

# Order by :

| Table Name | Column Name | Sort Order |
| --- | --- | --- |
| Purchasing.Suppliers | TotalPurchaseAmount | Descending |

# Problem solving Query:

USE WideWorldimporters; *-- MEDIUM QUERY*

GO

WITH SupplierPerformance AS (

SELECT

s.SupplierID,

s.SupplierName,

*AVG*(*DATEDIFF*(day, po.OrderDate, po.ExpectedDeliveryDate)) AS AverageLeadTimeDays,

*SUM*(pol.ExpectedUnitPricePerOuter \* pol.OrderedOuters) AS TotalPurchaseAmount

FROM

Purchasing.Suppliers s

INNER JOIN Purchasing.PurchaseOrders po ON s.SupplierID = po.SupplierID

INNER JOIN Purchasing.PurchaseOrderLines pol ON po.PurchaseOrderID = pol.PurchaseOrderID

WHERE

po.OrderDate BETWEEN '2013-01-01' AND '2013-12-31'

GROUP BY

s.SupplierID, s.SupplierName

)

SELECT

SupplierName,

AverageLeadTimeDays,

TotalPurchaseAmount

FROM

SupplierPerformance

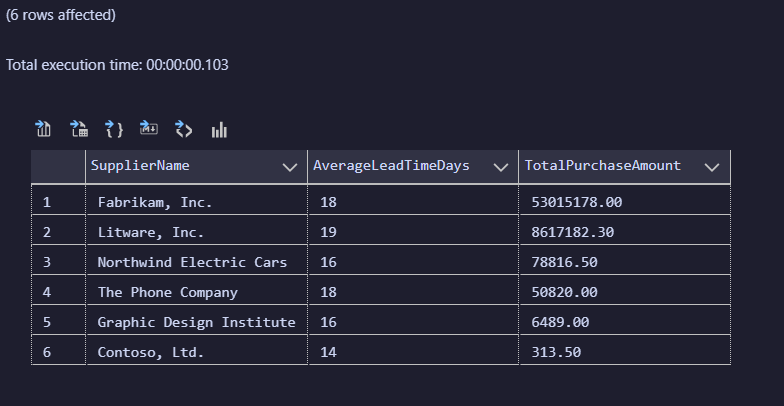
ORDER BY

TotalPurchaseAmount DESC

--FOR JSON PATH, ROOT('SupplierPerformance');

# Sample Relational Output with total number of rows returned:

# 



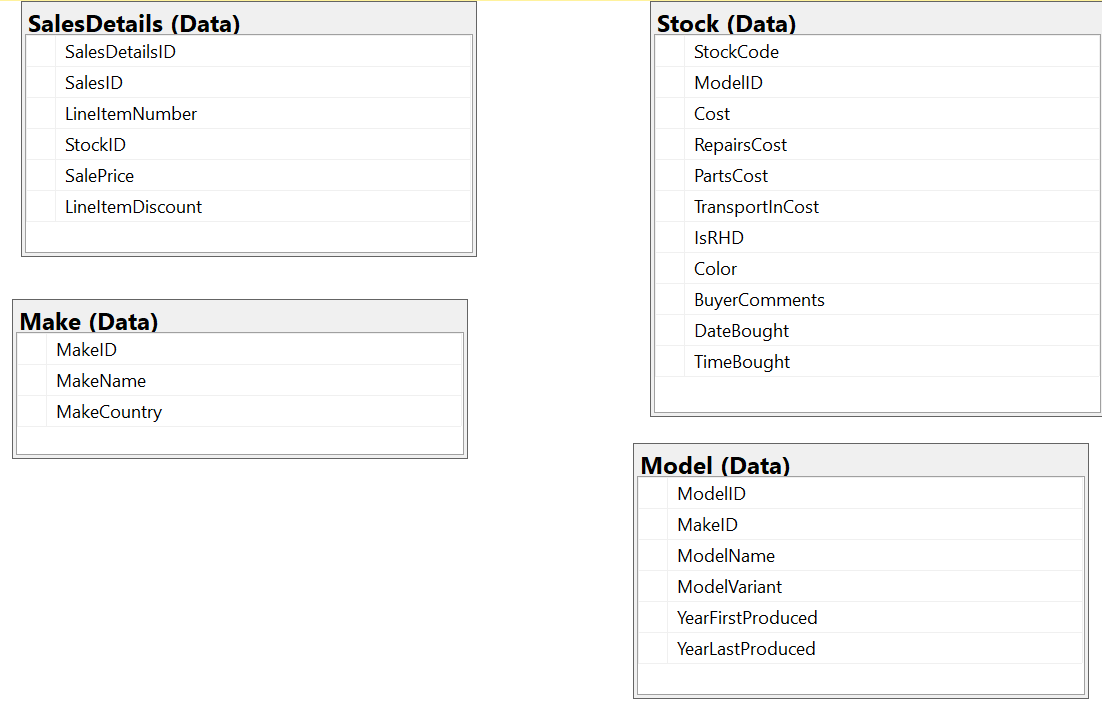
# Sample JSON Output with total number of rows returned:



# TOP QUERY (2)

USE PrestsigeCars :-

NOTE DATABASE SCRIPT DOES NOT DEFINE FOREIGN KEY AND PRIMARY KEY RELATIONSHIP



**Proposition:** Develop or update a function named dbo.CalculateTotalCost within the PrestigeCars database. This function computes the comprehensive cost for a vehicle, integrating the initial cost with the expenses incurred from repairs, parts, and transport.

# Columns from their respective tables in the select clause:

| Table Name | Column Name |
| --- | --- |
| Data.Stock | Cost , StockCode, ModelID |
| Data.Model | ModelID, ModelName, MakeID |
| Data.Make | MakeID, MakeName |
| Data.SalesDetails | SalePrice, StockID, SalesID |

# Order by :

| Table Name | Column Name | Sort Order |
| --- | --- | --- |
| SalesSummary CTE | TotalPurchaseAmount | Descending |

# Problem solving Query:

Use PrestigeCars *-- COMPLEX QUERY*

GO

CREATE OR ALTER FUNCTION dbo.CalculateTotalCost(

@Cost money,

@RepairsCost money,

@PartsCost money,

@TransportInCost money

)

RETURNS money

AS

BEGIN

RETURN @Cost + @RepairsCost + @PartsCost + @TransportInCost;

END;

GO

WITH CarSalesAnalysis AS (

SELECT

MK.MakeName,

MD.ModelName,

dbo.CalculateTotalCost(ST.Cost, ST.RepairsCost, ST.PartsCost, ST.TransportInCost) AS TotalCost,

SD.SalePrice

FROM Data.Stock ST

INNER JOIN Data.Model MD ON ST.ModelID = MD.ModelID

INNER JOIN Data.Make MK ON MD.MakeID = MK.MakeID

INNER JOIN Data.SalesDetails SD ON ST.StockCode = SD.StockID

),

SalesSummary AS (

SELECT

MakeName,

ModelName,

*SUM*(TotalCost) AS TotalCosts,

*SUM*(SalePrice) AS TotalSales,

*AVG*(SalePrice) AS AverageSalePrice,

*COUNT*(\*) AS NumberOfSales

FROM CarSalesAnalysis

GROUP BY MakeName, ModelName

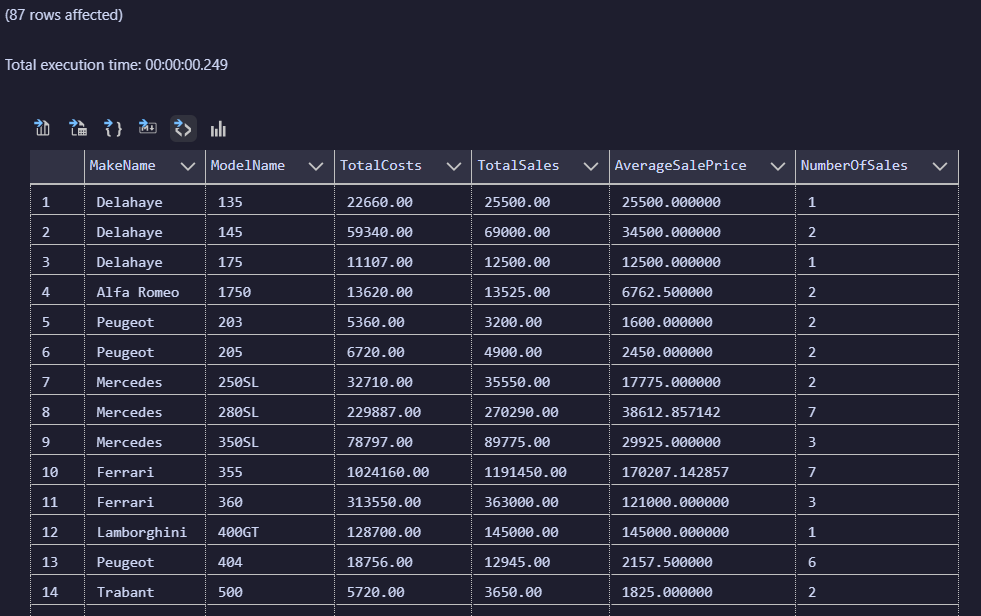
)

SELECT MakeName, ModelName, TotalCosts, TotalSales, AverageSalePrice, NumberOfSales

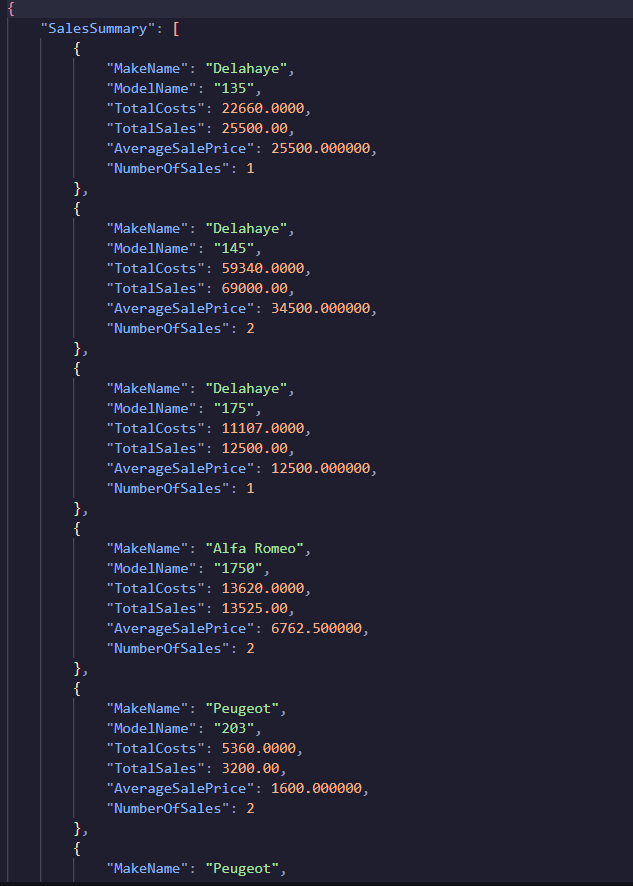
FROM SalesSummary

*--FOR JSON PATH, ROOT('SalesSummary');*

# Sample Relational Output with total number of rows returned:



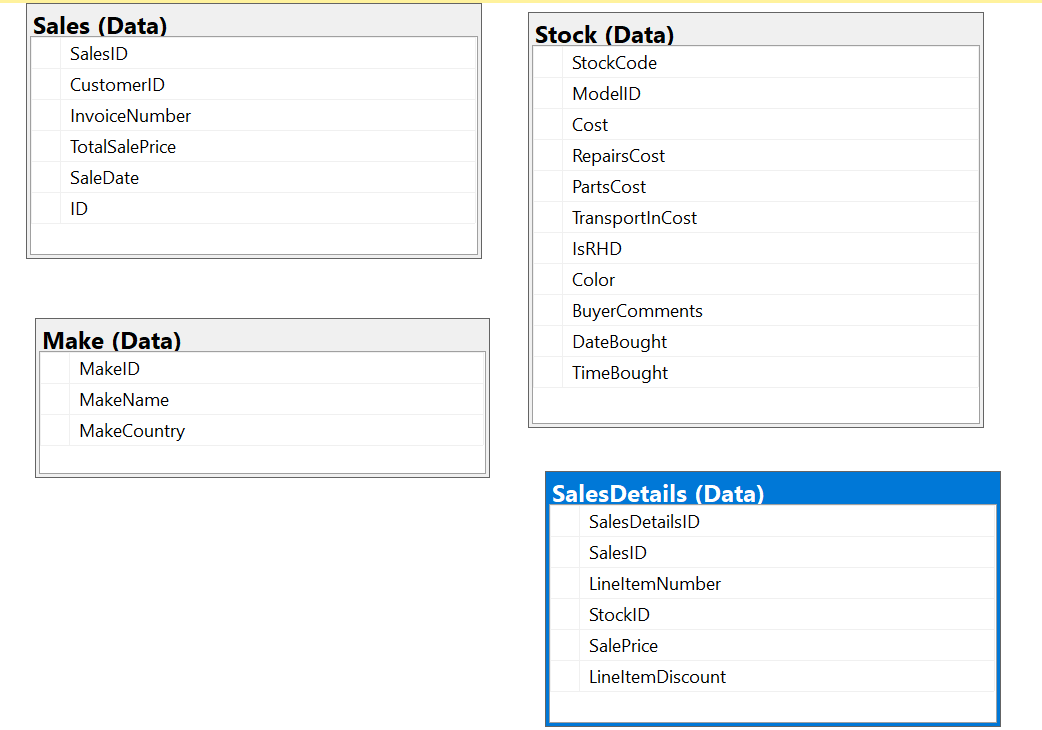
# Sample JSON Output with total number of rows returned:



# TOP QUERY (3):

Use PrestigeCars:

NOTE DATABASE SCRIPT DOES NOT DEFINE FOREIGN KEY AND PRIMARY KEY RELATIONSHIP



**Proposition**: Create or revise a function named dbo.CalculateTotalCost in the PrestigeCars database. This function determines the total cost for a vehicle by incorporating the base sale price and any additional charges.

# Columns from their respective tables in the select clause:

| Table Name | Column Name |
| --- | --- |
| Data.Sales (SA) | SalesID |
| Data.SalesDetails (SD) | SalePrice, SalesID, StockID |
| Data.Stock (ST) | StockCode, ModelID |
| Data.Make (MK) | MakeID |

# Order by :

| Table Name | Column Name | Sort Order |
| --- | --- | --- |
| N/A | N/A | N/A |

# Problem solving Query:

USE PrestigeCars;

GO

*-- CTE to aggregate sales by Make and Model*

WITH CarSalesSummary AS (

SELECT

MK.MakeName,

MD.ModelName,

*SUM*(SD.SalePrice) AS TotalSales,

*AVG*(SD.SalePrice) AS AverageSalePrice,

*COUNT*(SD.SalesID) AS NumberOfSales

FROM Data.Sales SA

INNER JOIN Data.SalesDetails SD ON SA.SalesID = SD.SalesID

INNER JOIN Data.Stock ST ON SD.StockID = ST.StockCode

INNER JOIN Data.Model MD ON ST.ModelID = MD.ModelID

INNER JOIN Data.Make MK ON MD.MakeID = MK.MakeID

GROUP BY MK.MakeName, MD.ModelName

)

*-- Select the data in relational format*

SELECT MakeName, ModelName, TotalSales, AverageSalePrice, NumberOfSales

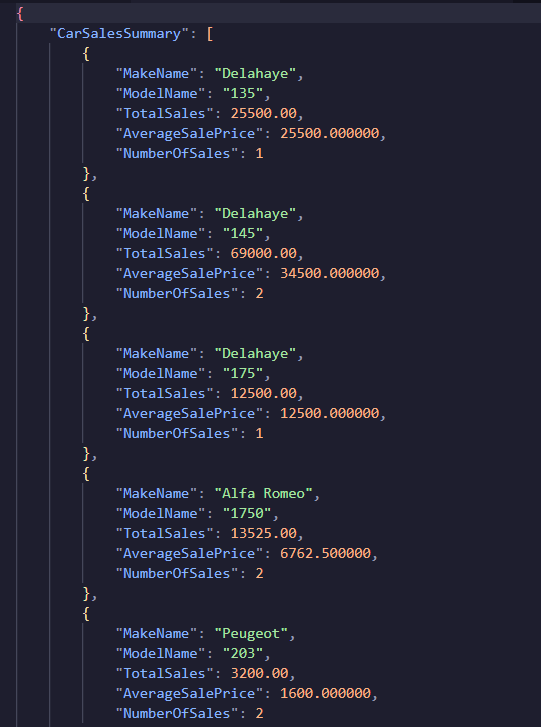
FROM CarSalesSummary

FOR JSON PATH, ROOT('CarSalesSummary');

# Sample Relational Output with total number of rows returned:



# Sample JSON Output with total number of rows returned:



# 

# WORST QUERY (1)

USE PrestigeCars

NOTE DATABASE SCRIPT DOES NOT DEFINE FOREIGN KEY AND PRIMARY KEY RELATIONSHIP

CANNOT AN ER DIAGRAM FROM A VIEW

# Columns from their respective tables in the select clause:

| Table Name | Column Name |
| --- | --- |
| [Data].[SalesByCountry]- VIEW | CountryName, SalePrice, LineItemDiscount, InvoiceNumber |

# Order by :

| Table Name | Column Name | Sort Order |
| --- | --- | --- |
| N/A | N/A | N/A |

# Problem solving Query:

USE PrestigeCars;

GO

*-- Query using the view [Data].[SalesByCountry] to summarize sales by country*

WITH SalesSummary AS (

SELECT

CountryName,

*SUM*(SalePrice - LineItemDiscount) AS TotalSalesValue,

*AVG*(SalePrice - LineItemDiscount) AS AverageSalePrice,

*COUNT*(DISTINCT InvoiceNumber) AS NumberOfTransactions

FROM [Data].[SalesByCountry]

GROUP BY CountryName

)

SELECT

CountryName,

TotalSalesValue,

AverageSalePrice,

NumberOfTransactions

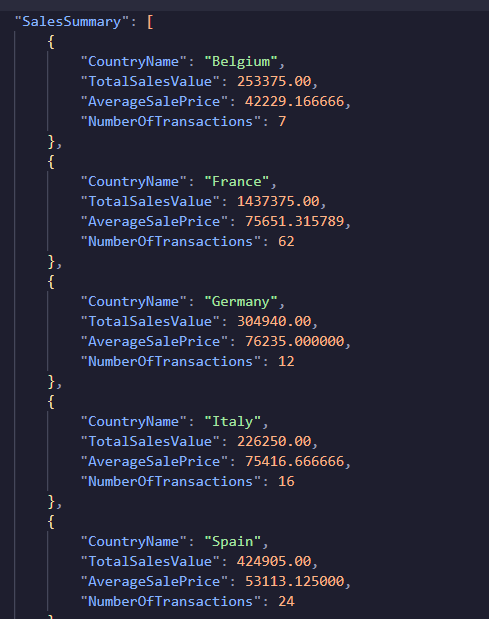
FROM SalesSummary

*-- FOR JSON PATH, ROOT('SalesSummary');*

# Sample Relational Output with total number of rows returned:

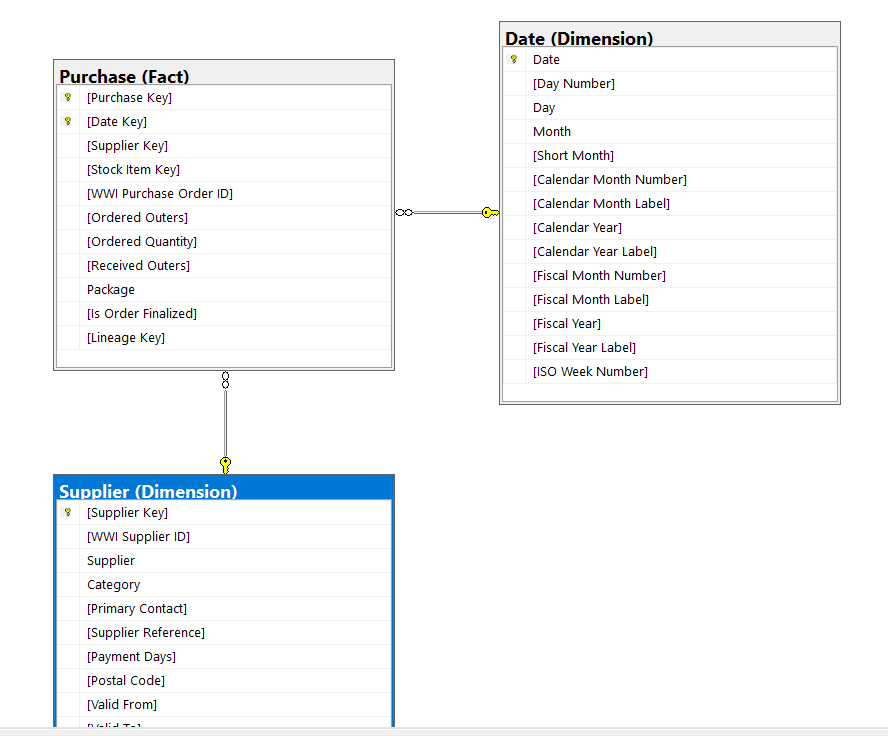
# 

# Sample JSON Output with total number of rows returned:



# WORST QUERY (2)

Use: WideWorldImportersDW



PROPOSITION: Formulate or update a query in the WideWorldImportersDW database to compute and summarize monthly purchases from suppliers. This involves calculating both the total and average quantity of items ordered from each supplier, segmented by month and year.

# 

# Columns from their respective tables in the select clause:

| Table Name | Column Name |
| --- | --- |
| [Data].[SalesByCountry]- VIEW | CountryName, SalePrice, LineItemDiscount, InvoiceNumber |
| Fact.Purchase | Date Key, Supplier Key |
| Dimension.Date | Date |
| Dimension.Supplier | Supplier Key |

# Order by :

| Table Name | Column Name | Sort Order |
| --- | --- | --- |
| SalesSummary CTE | TotalPurchaseAmount | Descending |

# Problem solving Query:

Use WideWorldImportersDW

;WITH MonthlySupplierPurchases AS (

SELECT

d.[Calendar Month Label] AS Month,

d.[Calendar Year] AS Year,

s.Supplier,

*SUM*(p.[Ordered Quantity]) AS TotalQuantity,

*AVG*(p.[Ordered Quantity]) AS AverageQuantity

FROM

Fact.Purchase AS p

JOIN Dimension.Date AS d ON p.[Date Key] = d.Date

JOIN Dimension.Supplier AS s ON p.[Supplier Key] = s.[Supplier Key]

GROUP BY

d.[Calendar Month Label],

d.[Calendar Year],

s.Supplier

)

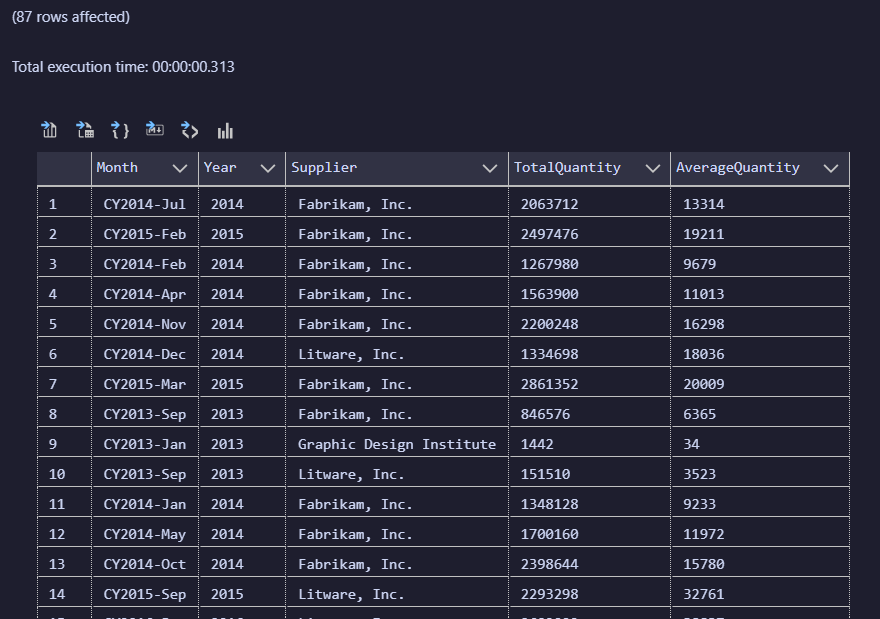
*-- Relational output*

SELECT Month, Year, Supplier, TotalQuantity, AverageQuantity

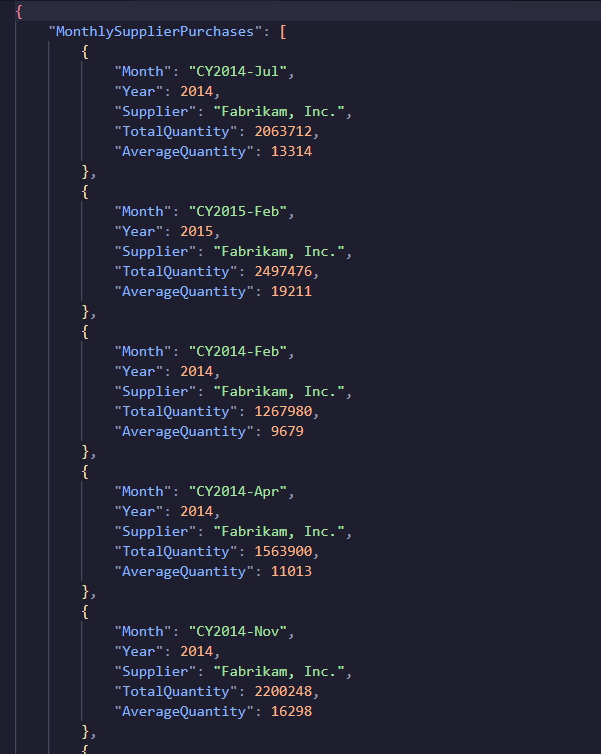
FROM MonthlySupplierPurchases

FOR JSON PATH, ROOT('MonthlySupplierPurchases');

# Sample Relational Output with total number of rows returned:

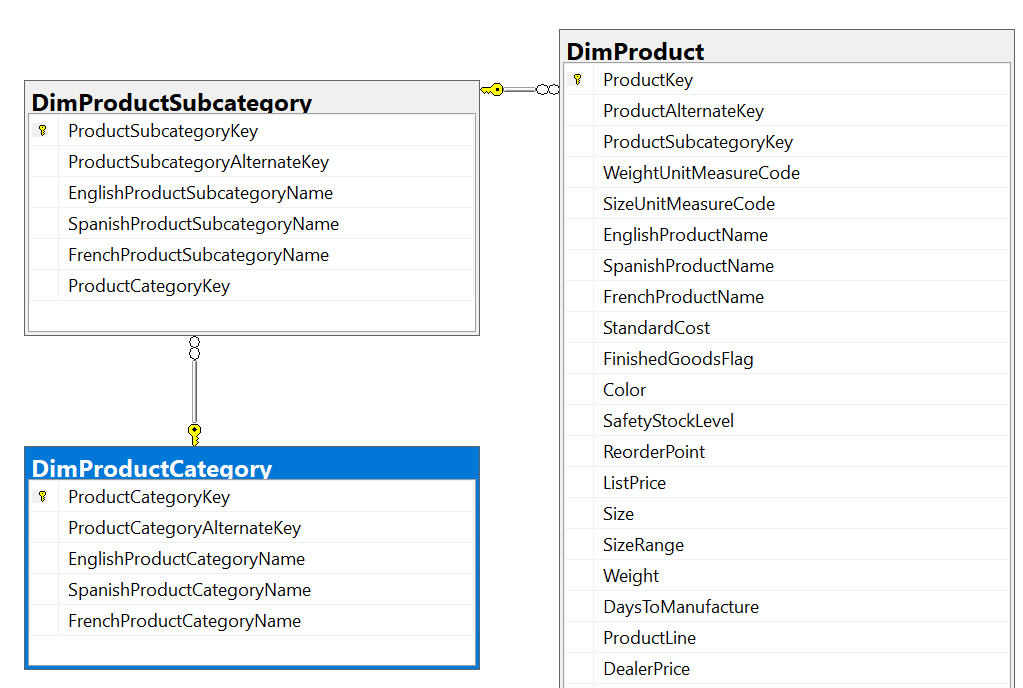


# Sample JSON Output with total number of rows returned:



# WORST QUERY (3)

USE AdeventureWorksDW2017



**Proposition:**

Utilizing a Common Table Expression (CTE) named ProductPriceSummary, the query systematically aggregates product pricing data from the dbo.DimProduct table, alongside hierarchical categorization from dbo.DimProductSubcategory and dbo.DimProductCategory tables. By filtering out products without a list price, it calculates the average list price within each product category and subcategory. The results are grouped accordingly, offering a refined view that facilitates understanding of average pricing across various product segments. This strategic grouping provides valuable insights for pricing analysis, marketing strategies, and product placement decisions.

# Columns from their respective tables in the select clause:

| Table Name | Column Name |
| --- | --- |
| dbo.DimProduct (p) | ListPrice, ProductSubcategoryKey |
| dbo.DimProductSubcategory (psc) | EnglishProductSubcategoryName, ProductCategoryKey |
| dbo.DimProductCategory (pc) | EnglishProductCategoryName |

# 

# Order by :

| Table Name | Column Name | Sort Order |
| --- | --- | --- |
| SalesSummary CTE | TotalPurchaseAmount | Descending |

# Problem solving Query:

Use AdventureWorksDW2017

*-- Calculating average list price by product category and subcategory*

;WITH ProductPriceSummary AS (

SELECT

pc.EnglishProductCategoryName AS CategoryName,

psc.EnglishProductSubcategoryName AS SubcategoryName,

*AVG*(p.ListPrice) AS AverageListPrice

FROM dbo.DimProduct AS p

INNER JOIN dbo.DimProductSubcategory AS psc ON p.ProductSubcategoryKey = psc.ProductSubcategoryKey

INNER JOIN dbo.DimProductCategory AS pc ON psc.ProductCategoryKey = pc.ProductCategoryKey

WHERE p.ListPrice > 0 *-- Excluding products with no list price*

GROUP BY pc.EnglishProductCategoryName, psc.EnglishProductSubcategoryName

)

*-- Relational output*

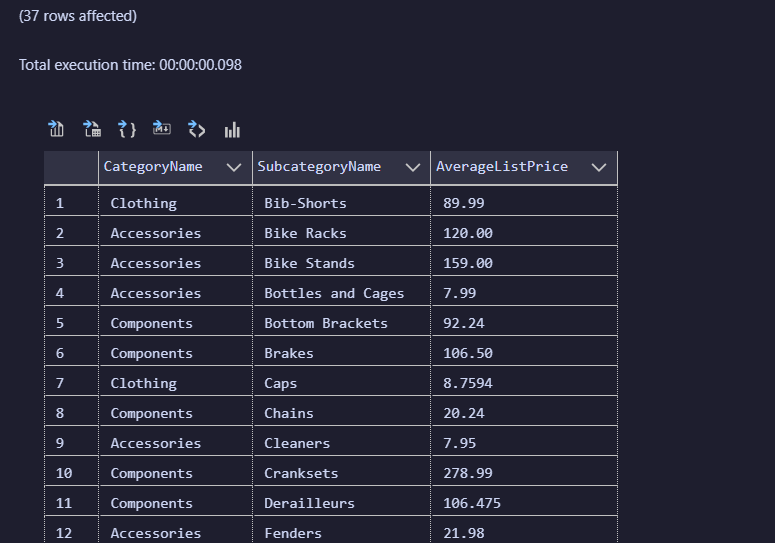
SELECT CategoryName, SubcategoryName, AverageListPrice

FROM ProductPriceSummary

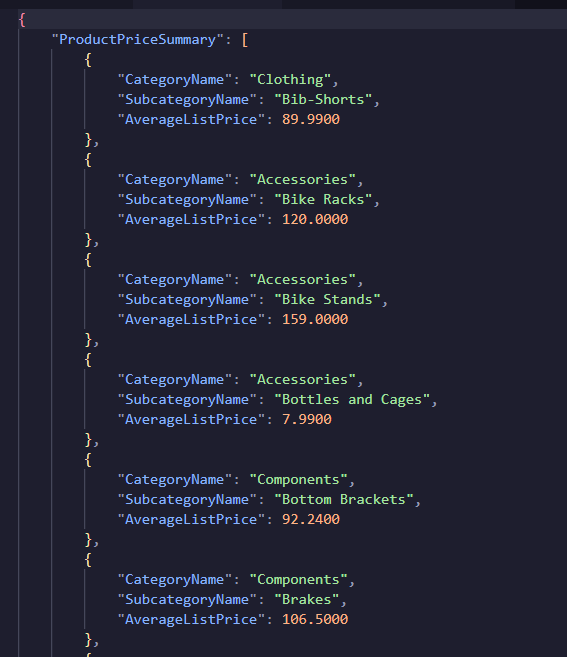
*-- JSON output*

FOR JSON PATH, ROOT('ProductPriceSummary');

# Sample Relational Output with total number of rows returned:

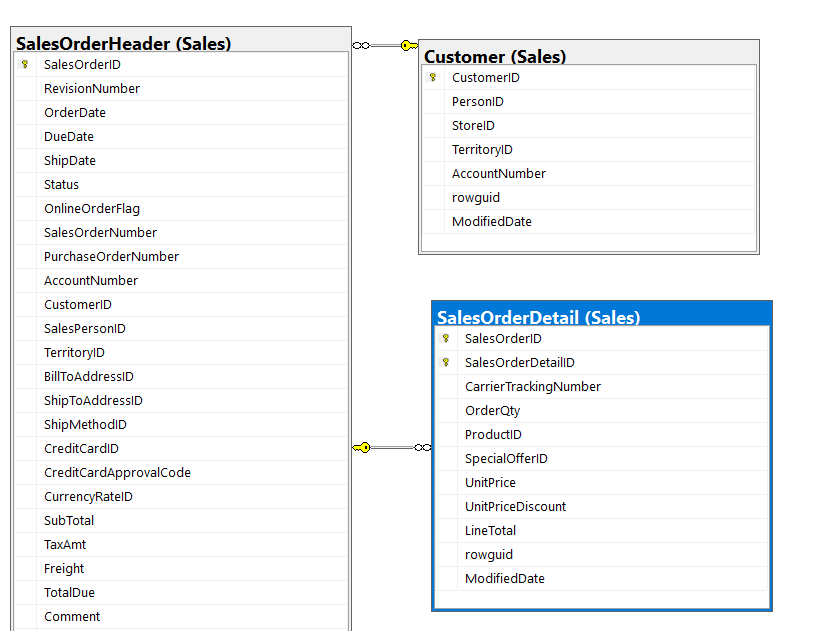


# Sample JSON Output with total number of rows returned:



# MEDIUM QUERY

USE AdeventureWorks2017



**Proposition:** Design or refine a query in the AdventureWorks2017 database for the purpose of summarizing sales data by customer. This summary includes the total number of orders, the total sales value, and the average value of orders placed by each customer.

# Columns from their respective tables in the select clause:

| Table Name | Column Name |
| --- | --- |
| Sales.Customer (c) | CustomerID |
| Sales.SalesOrderHeader (soh) | CustomerID, SalesOrderID |
| Sales.SalesOrderDetail (sod) | SalesOrderID, LineTotal |

# Order by :

| Table Name | Column Name | Sort Order |
| --- | --- | --- |
| Sales.Customer | CustomerID | Descending |

# 

# Problem solving Query:

Use AdventureWorks2017

;WITH CustomerSalesSummary AS (

SELECT

c.CustomerID,

*COUNT*(soh.SalesOrderID) AS TotalOrders,

*SUM*(sod.LineTotal) AS TotalSales,

*AVG*(sod.LineTotal) AS AverageOrderValue

FROM Sales.Customer c

INNER JOIN Sales.SalesOrderHeader soh ON c.CustomerID = soh.CustomerID

INNER JOIN Sales.SalesOrderDetail sod ON soh.SalesOrderID = sod.SalesOrderID

GROUP BY c.CustomerID

)

*-- Relational output*

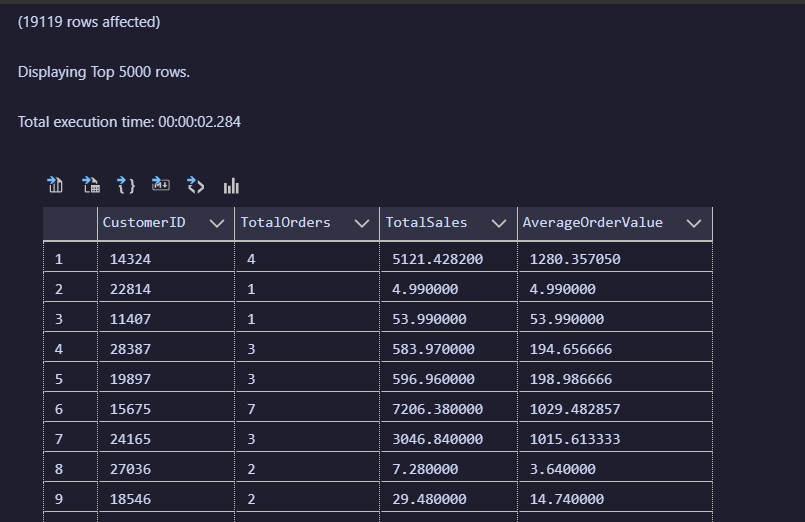
SELECT CustomerID, TotalOrders, TotalSales, AverageOrderValue

FROM CustomerSalesSummary

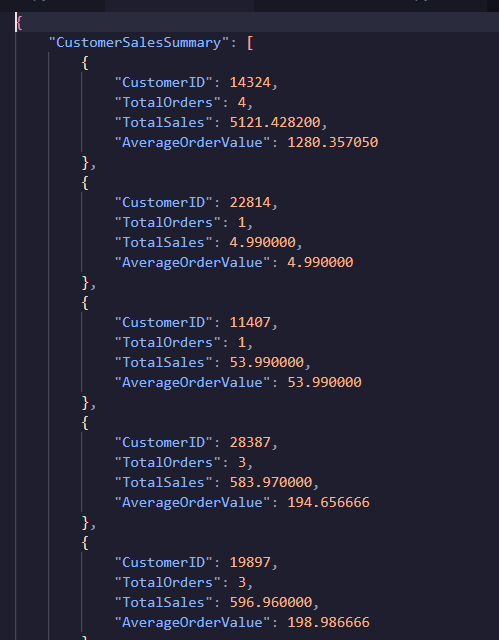
*-- JSON output*

FOR JSON PATH, ROOT('CustomerSalesSummary');

# Sample Relational Output with total number of rows returned:

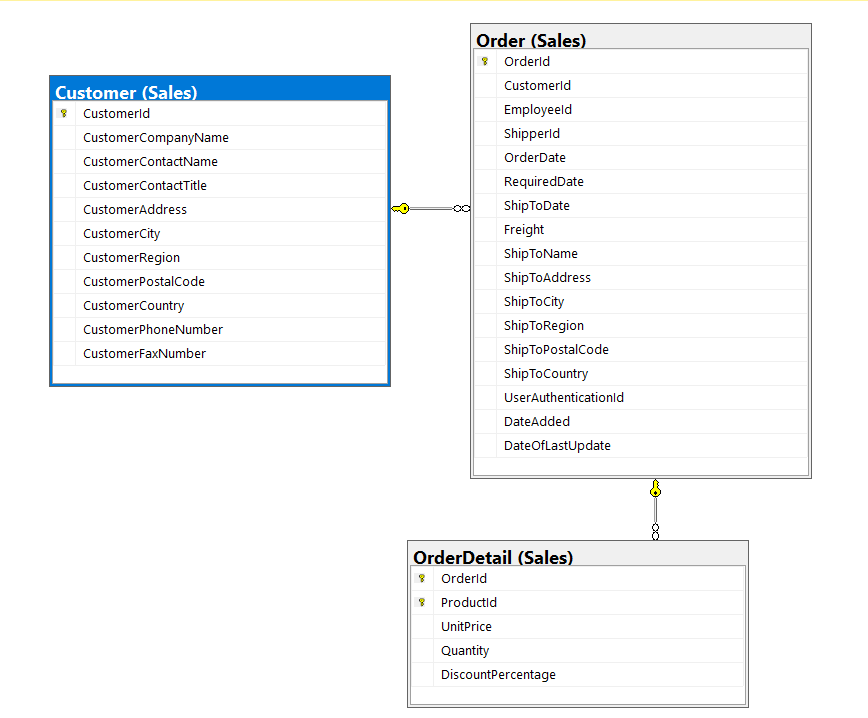


# Sample JSON Output with total number of rows returned:



# MEDIUM QUERY

Use Northwinds2022TSQLV7



**PROPOSITION**: Develop or enhance a query within the AdventureWorks2017 database to aggregate sales data by customer. This query focuses on computing the total number of orders, the total sales value, and the average order value for each customer.

# Columns from their respective tables in the select clause:

| Table Name | Column Name |
| --- | --- |
| Sales.Customer (c) | CustomerCompanyName |
| Sales.[Order] (o) | CustomerId, OrderId |
| Sales.OrderDetail (od) | OrderId, UnitPrice, Quantity |

# Order by :

| Table Name | Column Name | Sort Order |
| --- | --- | --- |
| N/A | N/A | N/A |

# Problem solving Query:

Use Northwinds2022TSQLV7

;WITH CustomerOrderSummary AS (

SELECT

c.CustomerCompanyName,

*COUNT*(DISTINCT o.OrderId) AS TotalOrders,

*SUM*(od.UnitPrice \* od.Quantity) AS TotalSalesValue,

*SUM*(od.Quantity) AS TotalProductsOrdered

FROM Sales.[Order] o

INNER JOIN Sales.Customer c ON o.CustomerId = c.CustomerId

INNER JOIN Sales.OrderDetail od ON o.OrderId = od.OrderId

GROUP BY c.CustomerCompanyName

)

*-- Relational output*

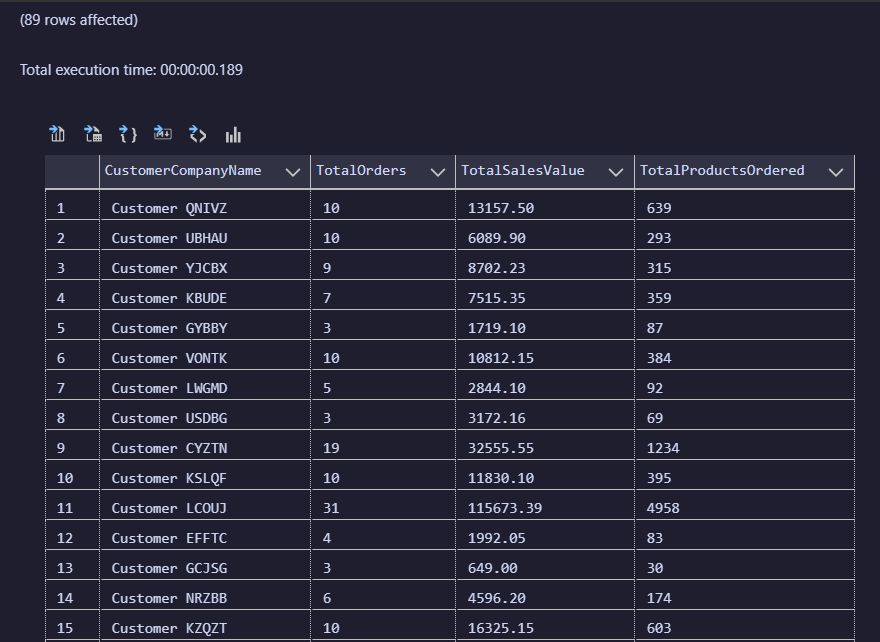
SELECT CustomerCompanyName, TotalOrders, TotalSalesValue, TotalProductsOrdered

FROM CustomerOrderSummary

*-- JSON output*

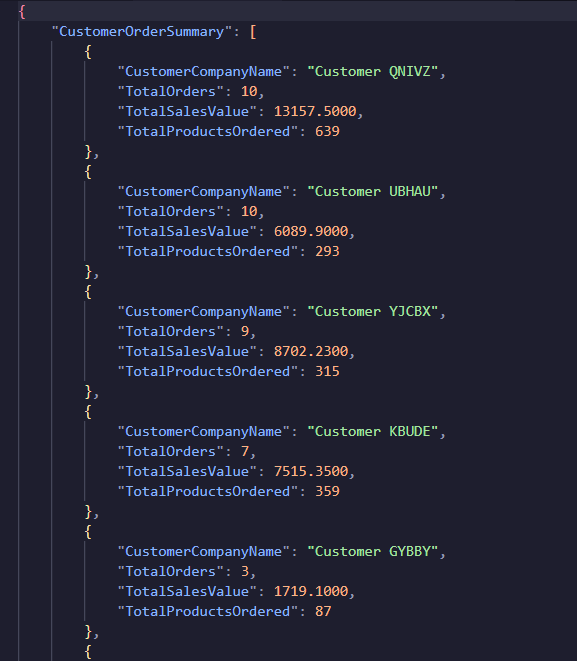
FOR JSON PATH, ROOT('CustomerOrderSummary');

# Sample Relational Output with total number of rows returned:



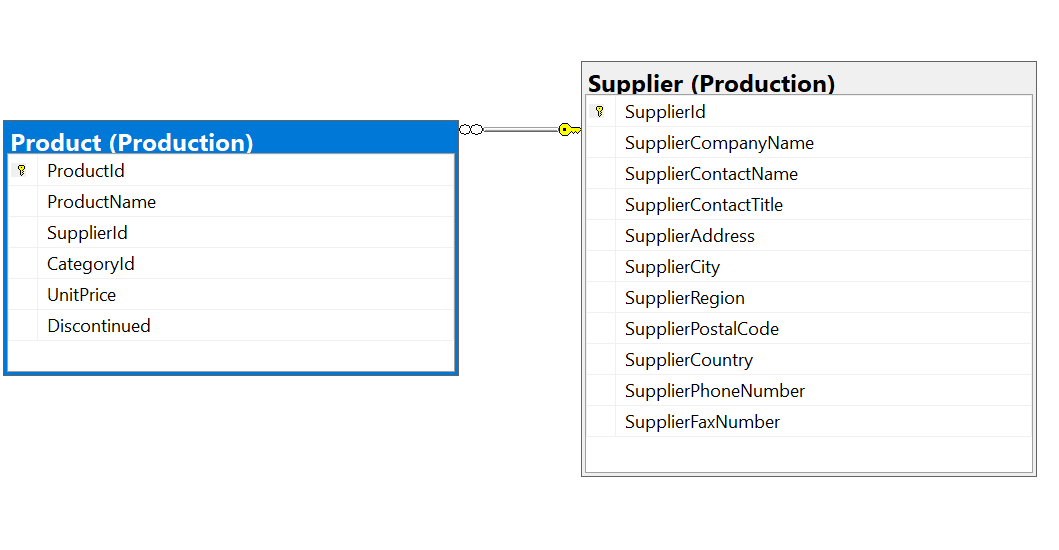
# 

# Sample JSON Output with total number of rows returned:



# MEDIUM QUERY

Use Northwinds2022TSQLV7



**PREPOSITION:** Design or modify a query in the Northwinds2022TSQLV7 database to compile a summary of orders by customer company. This query aims to detail the total number of distinct orders, the aggregate sales value, and the total quantity of products ordered, all categorized by customer company name.

# Columns from their respective tables in the select clause:

| Table Name | Column Name |
| --- | --- |
| Production.Supplier: | SupplierCompanyName |
| Production.Product | ProductId  SupplierId  UnitPrice |

# Order by :

| Table Name | Column Name | Sort Order |
| --- | --- | --- |
| N/A | N/A | N/A |

# Problem solving Query:

Use Northwinds2022TSQLV7

;WITH SupplierProductSummary AS (

SELECT

s.SupplierCompanyName,

*COUNT*(p.ProductId) AS TotalProducts,

*AVG*(p.UnitPrice) AS AverageUnitPrice

FROM Production.Product p

INNER JOIN Production.Supplier s ON p.SupplierId = s.SupplierId

GROUP BY s.SupplierCompanyName

)

*-- Relational output*

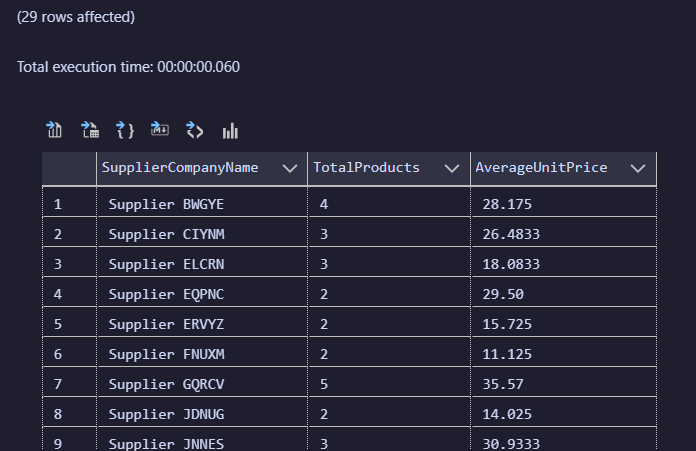
SELECT SupplierCompanyName, TotalProducts, AverageUnitPrice

FROM SupplierProductSummary

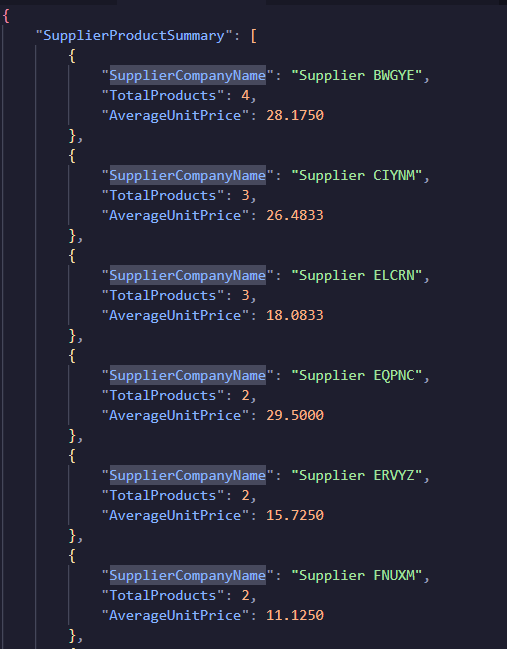
*-- -- JSON output*

FOR JSON PATH, ROOT('SupplierProductSummary');

# Sample Relational Output with total number of rows returned:

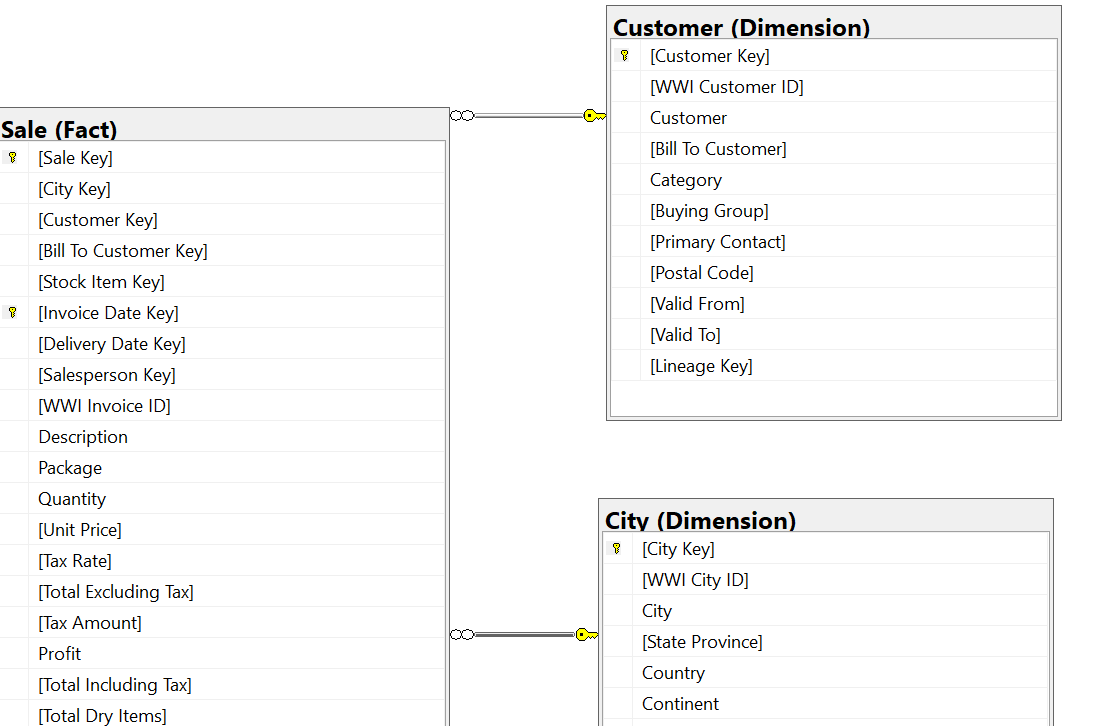


# Sample JSON Output with total number of rows returned:



# MEDIUM QUERY

Use WideWorldImportersDW



**PROPOSITION:**

Formulate or adjust a query in the WideWorldImportersDW database to synthesize sales data per customer, incorporating the city context. This endeavor aims to present both the total and average sales amounts, uniquely identifying each customer by their key and including the city for further geographical insights.

# Columns from their respective tables in the select clause:

| Table Name | Column Name |
| --- | --- |
| Dimension.Customer | [Customer Key], Customer |
| Fact.Sale | [Total Including Tax], [Customer Key], [City Key] |
| Dimension.City | City, [City Key] |

# 

# Order by :

| Table Name | Column Name | Sort Order |
| --- | --- | --- |
| Dimension.Customer | [Customer Key] | Asc |
| Dimension.Customer | City | Asc |

# 

# Problem solving Query:

Use WideWorldImportersDW

;WITH SalesByCustomer AS (

SELECT

c.[Customer Key],

c.Customer,

ci.City,

*SUM*(s.[Total Including Tax]) AS TotalSales,

*AVG*(s.[Total Including Tax]) AS AverageSaleAmount

FROM

Dimension.Customer AS c

JOIN Fact.Sale AS s ON c.[Customer Key] = s.[Customer Key]

JOIN Dimension.City AS ci ON s.[City Key] = ci.[City Key]

GROUP BY

c.[Customer Key],

c.Customer,

ci.City

)

*-- Relational output*

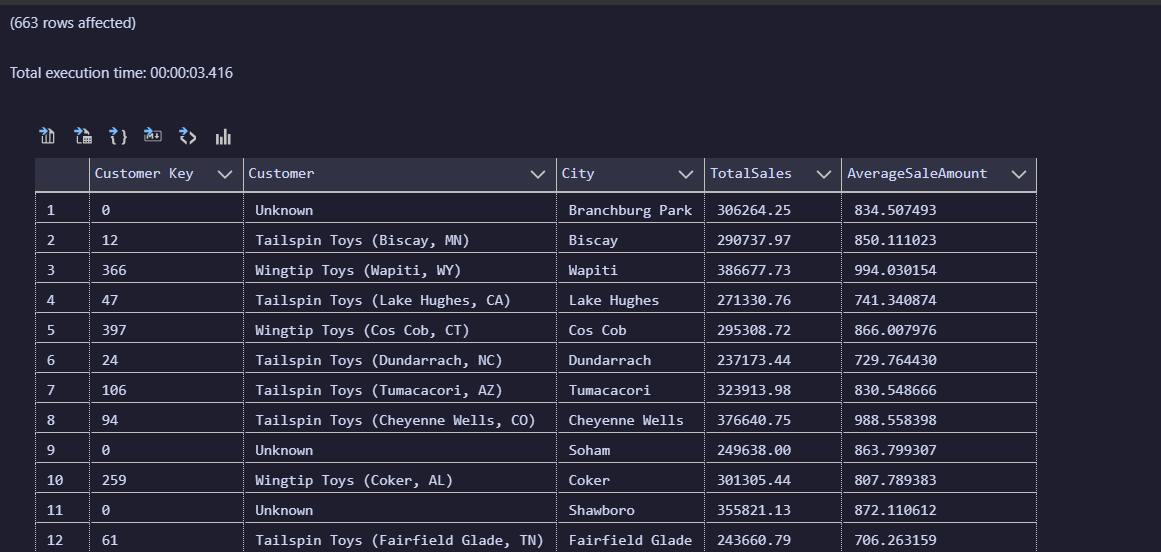
SELECT [Customer Key], Customer, City, TotalSales, AverageSaleAmount

FROM SalesByCustomer

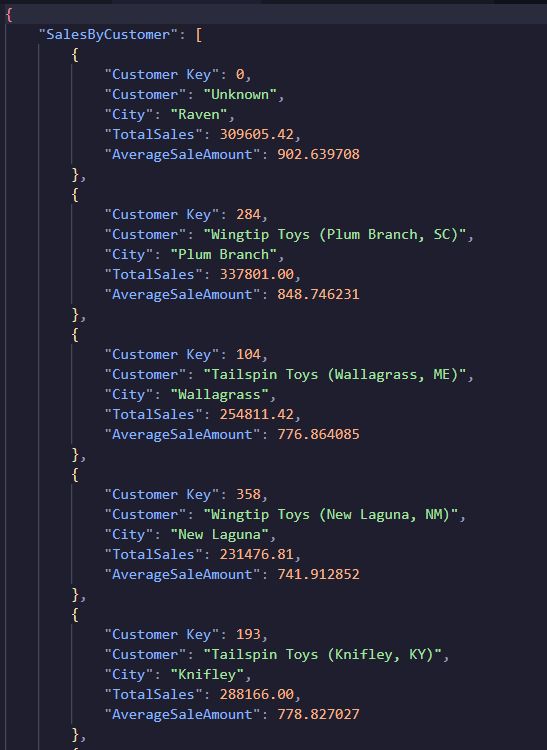
*-- JSON output*

FOR JSON PATH, ROOT('SalesByCustomer');

# Sample Relational Output with total number of rows returned:

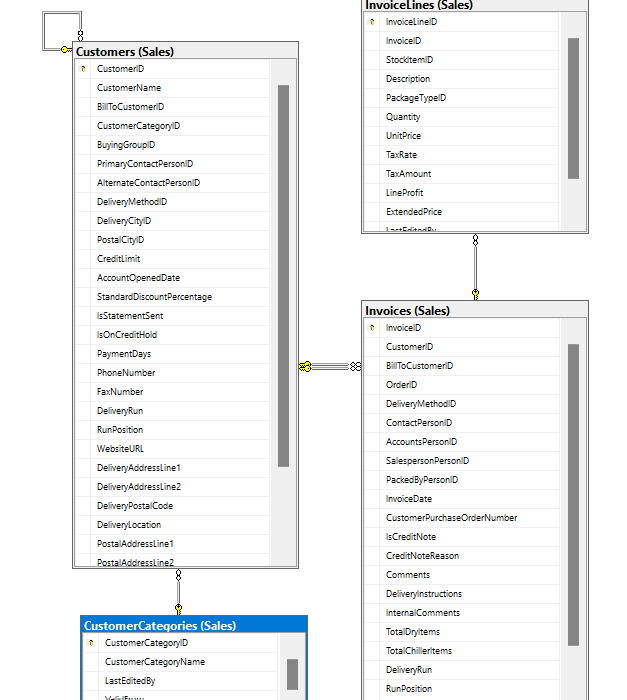


# Sample JSON Output with total number of rows returned:



# MEDIUM QUERY

Use WorldWideImporters



**PROPOSITION**: Create or update a view named Sales.CustomerSalesSummary to store aggregated sales data for customer categories based on their purchases. Then, select and display CustomerCategoryName, TotalSales, and AverageSalePerInvoice for each customer category from the Sales.CustomerSalesSummary view.

# Columns from their respective tables in the select clause:

| Table Name | Column Name |
| --- | --- |
| Sales.Customers table | CustomerID  CustomerCategoryID |
| Sales.Invoices table | CustomerID (for joining with Sales.Customers)  InvoiceID  InvoiceDate |
| Sales.InvoiceLines table | InvoiceID (for joining with Sales.Invoices)  InvoiceLineID  ExtendedPrice |
| Sales.CustomerCategories | CustomerCategoryID  CustomerCategoryName |

# 

# Order by :

| Table Name | Column Name | Sort Order |
| --- | --- | --- |
| Sales.InvoiceLines | TotalSales | Desc |

# 

# Problem solving Query:

USE WideWorldimporters; *-- Medium*

GO

WITH CustomerSales AS (

SELECT

c.CustomerID,

c.CustomerCategoryID,

i.InvoiceDate,

il.InvoiceLineID,

il.ExtendedPrice

FROM

Sales.Customers c

JOIN Sales.Invoices i ON c.CustomerID = i.CustomerID

JOIN Sales.InvoiceLines il ON i.InvoiceID = il.InvoiceID

WHERE

i.InvoiceDate BETWEEN '2014-01-01' AND '2014-12-31'

),

AggregatedSales AS (

SELECT

CustomerCategoryID,

*SUM*(ExtendedPrice) AS TotalSales,

*AVG*(ExtendedPrice) AS AverageSalePerInvoice

FROM

CustomerSales

GROUP BY

CustomerCategoryID

)

SELECT

cc.CustomerCategoryName,

asales.TotalSales,

asales.AverageSalePerInvoice

FROM

AggregatedSales asales

JOIN Sales.CustomerCategories cc ON asales.CustomerCategoryID = cc.CustomerCategoryID

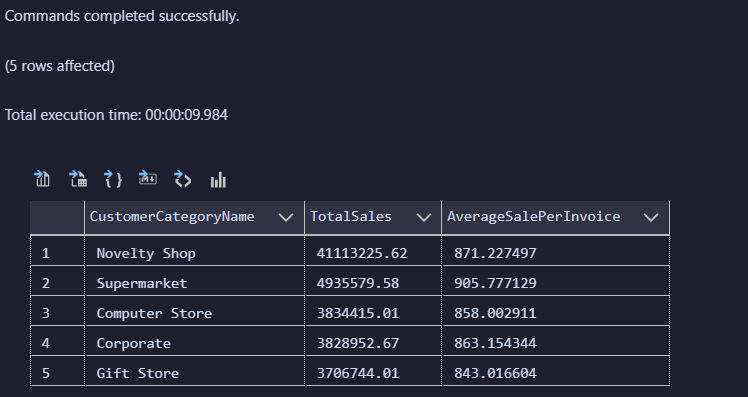
ORDER BY

TotalSales DESC

*-- JSON output*

FOR JSON PATH, ROOT('CustomerSales');

# Sample Relational Output with total number of rows returned:

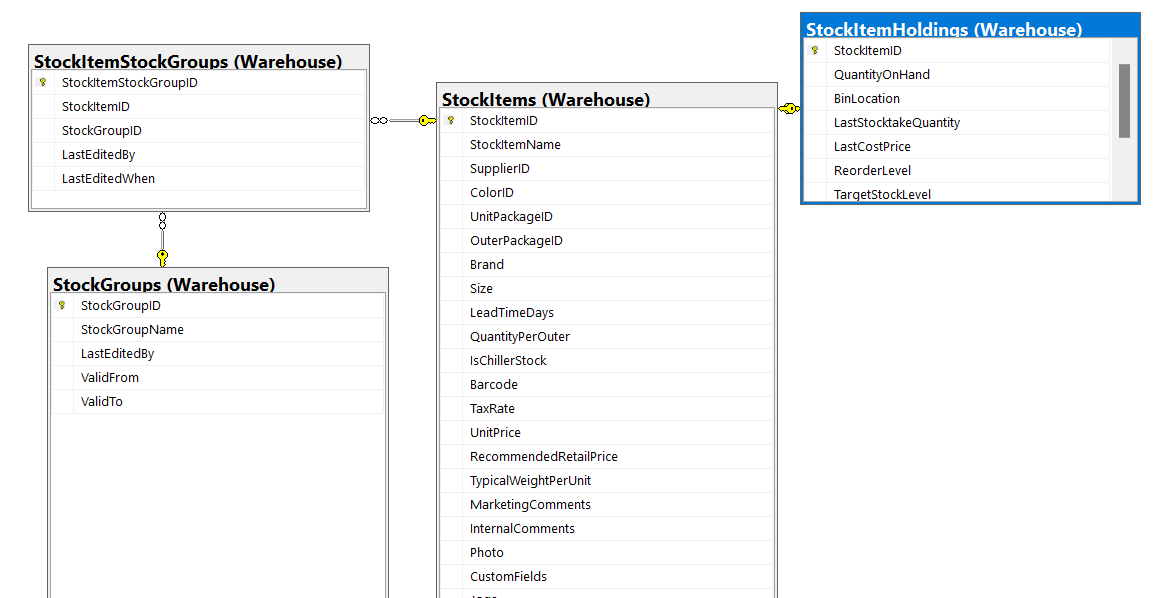


# Sample JSON Output with total number of rows returned:



# MEDIUM QUERIES

Use WideWorldimporters



**Preposition:** Create or update a view named Warehouse.StockSummary to store summarized information about stock items including their names, quantities on hand, and associated stock group names. Then, select and display StockItemName, QuantityOnHand, and StockGroupName for each stock item from the Warehouse.StockSummary view.

# Columns from their respective tables in the select clause:

| Table Name | Column Name |
| --- | --- |
| Warehouse.StockItemHoldings | StockItemID  QuantityOnHand |
| Warehouse.StockItems | StockItemID  StockItemName |
| Warehouse.StockItemStockGroups | StockItemID  StockGroupID |
| Warehouse.StockGroups | StockGroupID  StockGroupName |

# 

# Order by :

| Table Name | Column Name | Sort Order |
| --- | --- | --- |
| Warehouse.StockGroups | StockGroupName | Ascending |
| Warehouse.StockItems | StockItemName | Ascending |

# 

# Problem solving Query:

USE WideWorldimporters;

GO

WITH StockSummary AS (

SELECT

si.StockItemID,

si.StockItemName,

sish.QuantityOnHand,

sig.StockGroupName

FROM

Warehouse.StockItemHoldings sish

JOIN Warehouse.StockItems si ON sish.StockItemID = si.StockItemID

JOIN Warehouse.StockItemStockGroups sisg ON si.StockItemID = sisg.StockItemID

JOIN Warehouse.StockGroups sig ON sisg.StockGroupID = sig.StockGroupID

)

SELECT

StockItemName,

QuantityOnHand,

StockGroupName

FROM

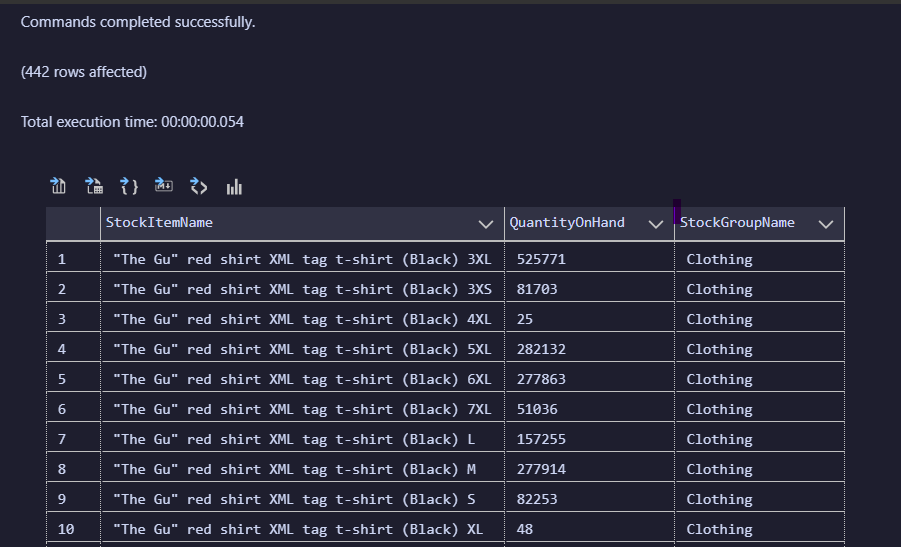
StockSummary

ORDER BY

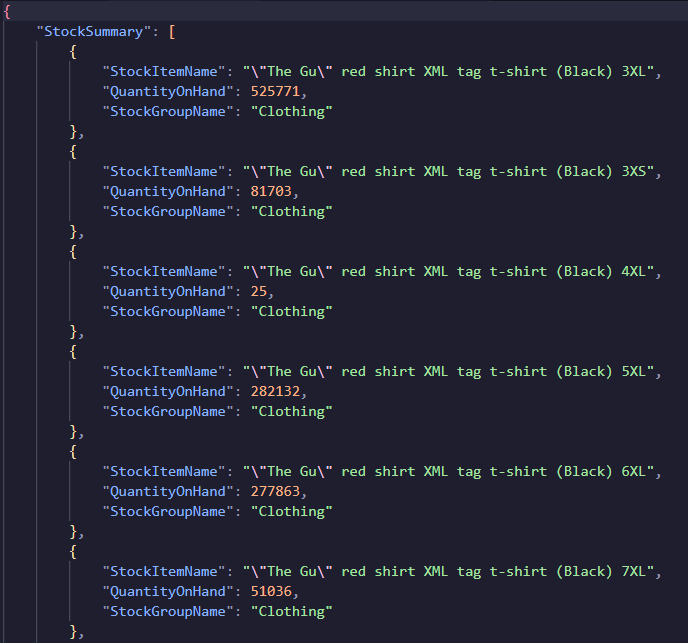
StockGroupName, StockItemName

FOR JSON PATH, ROOT('StockSummary');

# Sample Relational Output with total number of rows returned:

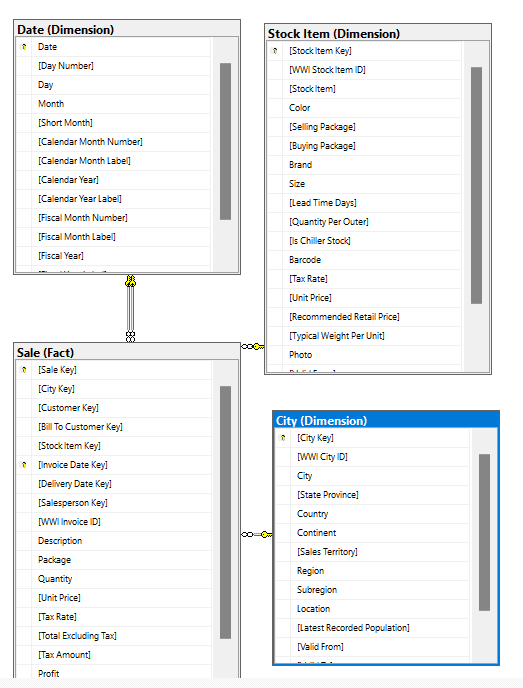


# Sample JSON Output with total number of rows returned:



# COMPLEX QUERY

Use WideWorldimportersDW



**Proposition:** Create or update a view named Sales.RegionalTopSellingProducts to store summarized sales data by region and top-selling products for the year 2013. Then, select and display Region, StockItem, and TotalSales for the top 5 selling products in each region from the Sales.RegionalTopSellingProducts view.

# Columns from their respective tables in the select clause:

| Table Name | Column Name |
| --- | --- |
| Fact.Sale | [Total Including Tax]  [Stock Item Key]  [City Key]  [Invoice Date Key] |
| Dimension.[Stock Item] | [Stock Item] (referred to as [Stock Item] in the query)  [Stock Item Key] |
| Dimension.City | Region  [City Key] |
| Dimension.Date | Date (used for joining with [Invoice Date Key])  [Calendar Year] |

# Order by :

| Table Name | Column Name | Sort Order |
| --- | --- | --- |
| Dimension.City | Region | Ascending |
| RankedRegionalSales CTE, TotalSales | SalesRank | Ascending |

# Problem solving Query:

USE WideWorldimportersDW; *-- COMPLEX*

GO

*-- Define a CTE to summarize sales data by region and product category in 2020*

WITH RegionalSalesData AS (

SELECT

ci.Region,

si.[Stock Item],

*SUM*(fs.[Total Including Tax]) AS TotalSales

FROM

Fact.Sale fs

INNER JOIN Dimension.[Stock Item] si ON fs.[Stock Item Key] = si.[Stock Item Key]

INNER JOIN Dimension.City ci ON fs.[City Key] = ci.[City Key]

INNER JOIN Dimension.Date d ON fs.[Invoice Date Key] = d.Date

WHERE

d.[Calendar Year] = 2013

GROUP BY

ci.Region,

si.[Stock Item]

),

*-- Rank sales data within each region to identify top-selling products*

RankedRegionalSales AS (

SELECT

Region,

[Stock Item],

TotalSales,

*RANK*() OVER (PARTITION BY Region ORDER BY TotalSales DESC) AS SalesRank

FROM

RegionalSalesData

)

*-- Select the top 5 selling products in each region*

SELECT

Region,

[Stock Item],

TotalSales

FROM

RankedRegionalSales

WHERE

SalesRank <= 5

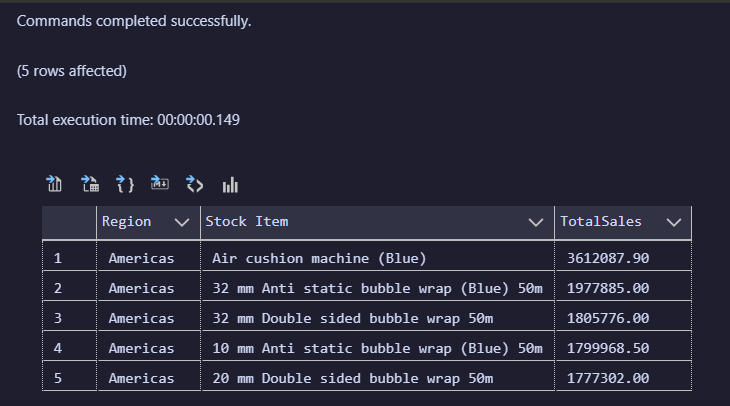
ORDER BY

Region,

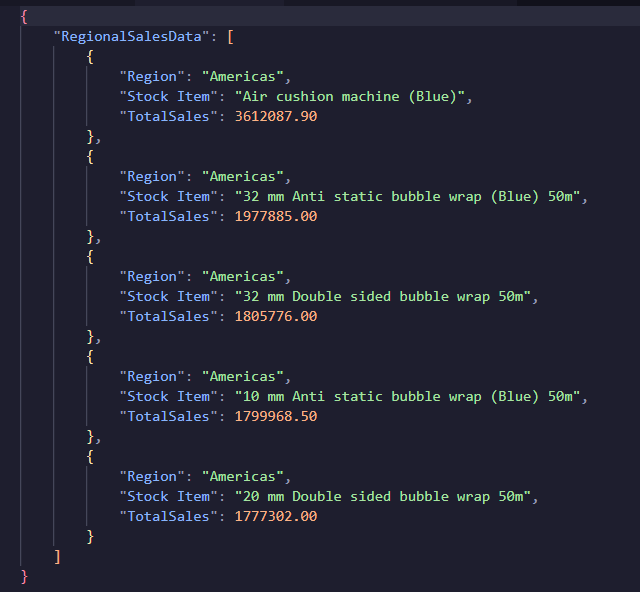
SalesRank

FOR JSON PATH, ROOT('RegionalSalesData');

# Sample Relational Output with total number of rows returned:



# Sample JSON Output with total number of rows returned:



# COMPLEX QUERY

Use WideWorldImporters

# 

**Proposition**: Establish or revise a function called dbo.GetTotalInvoiceTax within the WideWorldImporters database. The function is designed to compute the total tax amount for a given invoice and return it as a decimal value with precision up to 18 digits and 2 decimal places. The function queries the Sales.InvoiceLines table, sums up the TaxAmount column values where the InvoiceID matches the provided @InvoiceID parameter, and stores the result in a variable named @TotalTax. This computed total tax amount is then returned by the function.

# Columns from their respective tables in the select clause:

| Table Name | Column Name |
| --- | --- |
| Sales.Invoices | TaxAmount  InvoiceID  InvoiceID  CustomerID  InvoiceDate |
| Sales.InvoiceLines | ExtendedPrice  InvoiceLineID  InvoiceID (used for joining with Sales.Invoices) |
| Warehouse.StockItemTransactions | StockItemID  Quantity  TransactionOccurredWhen  InvoiceID |

# 

# Order by :

| Table Name | Column Name | Sort Order |
| --- | --- | --- |
| Sales.Invoices | InvoiceDate | DESC |
| Sales.Invoices | InvoiceID | ASC |

# 

# 

USE WideWorldImporters *--HARD*

GO

CREATE OR ALTER FUNCTION dbo.GetTotalInvoiceTax(@InvoiceID INT)

RETURNS DECIMAL(18,2)

AS

BEGIN

DECLARE @TotalTax DECIMAL(18,2);

SELECT @TotalTax = *SUM*(TaxAmount) FROM Sales.InvoiceLines WHERE InvoiceID = @InvoiceID;

RETURN @TotalTax;

END;

GO

;WITH InvoiceSummary AS (

SELECT

i.InvoiceID,

i.CustomerID,

i.InvoiceDate,

dbo.GetTotalInvoiceTax(i.InvoiceID) AS TotalTax,

*SUM*(il.ExtendedPrice) AS TotalExtendedPrice,

*COUNT*(il.InvoiceLineID) AS NumberOfItems

FROM Sales.Invoices i

INNER JOIN Sales.InvoiceLines il ON i.InvoiceID = il.InvoiceID

GROUP BY i.InvoiceID, i.CustomerID, i.InvoiceDate

),

StockTransactions AS (

SELECT

sit.StockItemID,

sit.Quantity,

sit.TransactionOccurredWhen,

sit.InvoiceID

FROM Warehouse.StockItemTransactions sit

WHERE sit.InvoiceID IS NOT NULL

)

SELECT

ISum.InvoiceID,

ISum.CustomerID,

ISum.InvoiceDate,

ISum.TotalTax,

ISum.TotalExtendedPrice,

ISum.NumberOfItems,

ST.Quantity AS TransactionQuantity,

ST.TransactionOccurredWhen

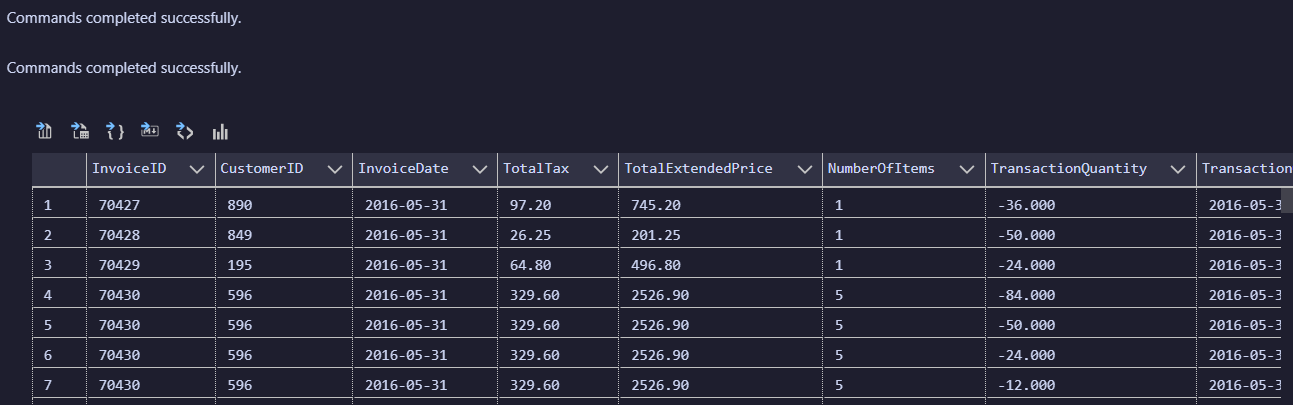
FROM InvoiceSummary ISum

INNER JOIN StockTransactions ST ON ISum.InvoiceID = ST.InvoiceID

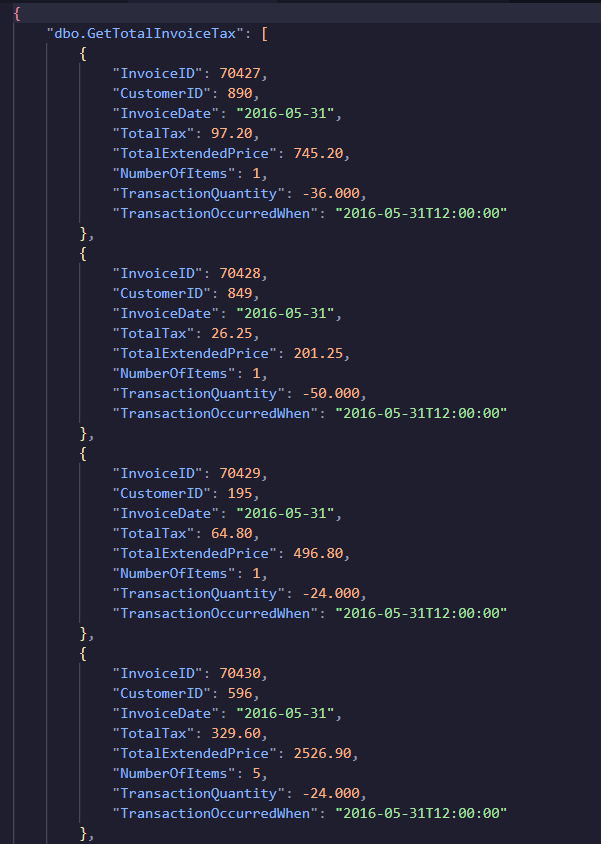
ORDER BY ISum.InvoiceDate DESC, ISum.InvoiceID

FOR JSON PATH, ROOT('dbo.GetTotalInvoiceTax');

# Sample Relational Output with total number of rows returned:



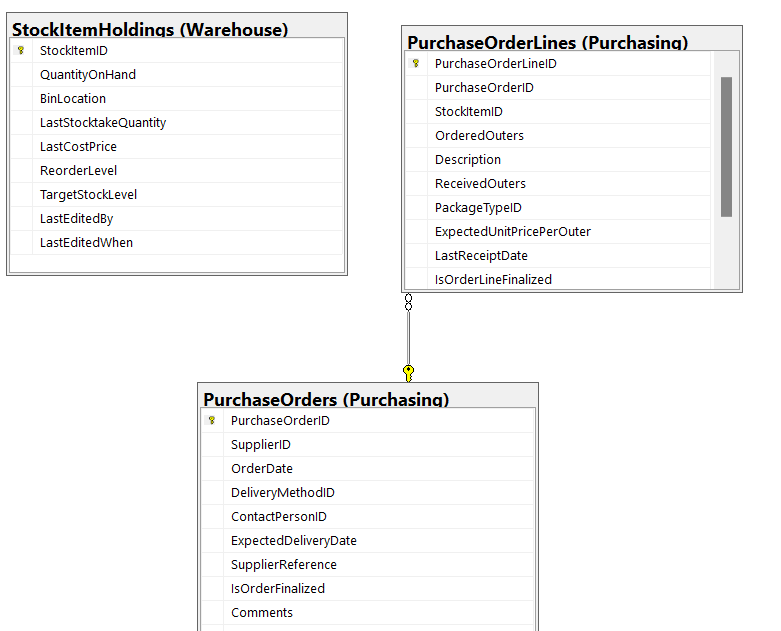
# Sample JSON Output with total number of rows returned:



# COMPLEX QUERY

USE WideWorldImporters

# Columns from their respective tables in the select clause:



**Proposition:** Develop or update a function named dbo.AvgCostPerItem in the WideWorldImporters database. This function is designed to compute the average cost per item for a specified purchase order.

# Columns from their respective tables in the select clause:

| Table Name | Column Name |
| --- | --- |
| Purchasing.PurchaseOrderLines  PurchaseOrderDetails CTE | PurchaseOrderID  StockItemID  OrderedOuters  ReceivedOuters |
| Warehouse.StockItemHoldings  PurchaseOrderDetails CTE | StockItemID  QuantityOnHand  LastCostPrice |
| Purchasing.PurchaseOrders | PurchaseOrderID  SupplierID  OrderDate |

# 

# Order by :

| Table Name | Column Name | Sort Order |
| --- | --- | --- |
| Purchasing.PurchaseOrders | OrderDate | DESC |

# 

# Problem solving Query:

USE WideWorldImporters; *--HARD*

GO

CREATE OR ALTER FUNCTION dbo.AvgCostPerItem(@PurchaseOrderID INT)

RETURNS DECIMAL(18,2)

AS

BEGIN

RETURN (

SELECT *AVG*(ExpectedUnitPricePerOuter)

FROM Purchasing.PurchaseOrderLines

WHERE PurchaseOrderID = @PurchaseOrderID

);

END;

GO

;WITH PurchaseOrderDetails AS (

SELECT

pol.PurchaseOrderID,

pol.StockItemID,

*SUM*(pol.OrderedOuters) AS TotalOrderedOuters,

*SUM*(pol.ReceivedOuters) AS TotalReceivedOuters

FROM Purchasing.PurchaseOrderLines pol

GROUP BY pol.PurchaseOrderID, pol.StockItemID

), StockSummary AS (

SELECT

sih.StockItemID,

sih.QuantityOnHand,

*AVG*(sih.LastCostPrice) AS AvgLastCostPrice *-- Assuming AVG is meaningful here*

FROM Warehouse.StockItemHoldings sih

GROUP BY sih.StockItemID, sih.QuantityOnHand

)

SELECT

po.PurchaseOrderID,

po.SupplierID,

po.OrderDate,

POD.TotalOrderedOuters,

POD.TotalReceivedOuters,

dbo.AvgCostPerItem(po.PurchaseOrderID) AS AvgCostPerItem,

SS.QuantityOnHand,

SS.AvgLastCostPrice

FROM Purchasing.PurchaseOrders po

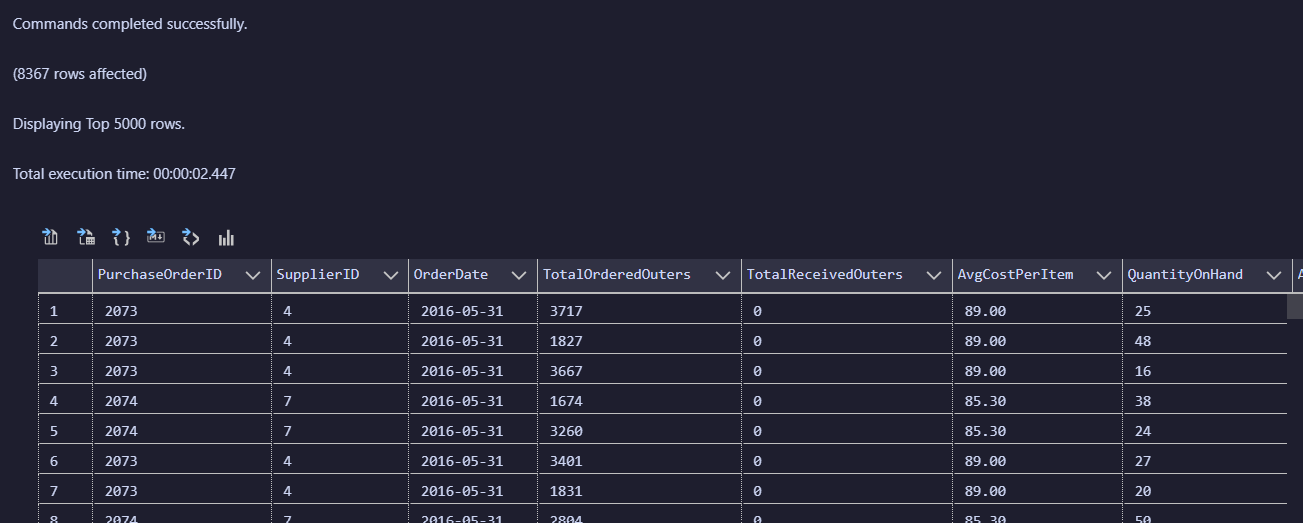
INNER JOIN PurchaseOrderDetails POD ON po.PurchaseOrderID = POD.PurchaseOrderID

LEFT JOIN StockSummary SS ON POD.StockItemID = SS.StockItemID

ORDER BY po.OrderDate DESC

FOR JSON PATH, ROOT('dbo.AvgCostPerItem');

# Sample Relational Output with total number of rows returned:



# Sample JSON Output with total number of rows returned:

